29-战术设计：如何像写故事一样找出模型？

你好，我是郑晔！

在上一讲中，我们讲了 DDD 中的战略设计，学习如何将识别出来的不同模型放到不同的限界上下文中。那么，接下来，我们就该做更具体的工作了，也就是如何设计模型。在 DDD 中，把具体的模型找出来的做法有一个更响亮的名字：战术设计。

战术设计同样也包含了很多概念，比如，实体、值对象、聚合、领域服务、应用服务等等。有这么多概念，我们该如何区分和理解他们呢？我们同样需要一根主线。

其实，我们可以把战术设计理解成写一个故事。你知道怎样去写个故事吗？写故事通常都是有一定套路的。我们要先构建好故事的背景，然后，要设定不同的角色，接下来，创建角色之间的关系，最后，我们要安排人物之间互动起来，形成故事。

对于战术设计而言，故事的背景就是我们面对的领域问题，剩下的就是我们在这个故事背景下，要找出不同的角色，找出角色之间的关系，让它们互动起来，这样，我们就有了故事，也完成了战术设计。

接下来，我们就来看看，战术设计这个故事模板，我们应该怎么填？

角色：实体、值对象

我们的首要任务就是设计角色，在战术设计中，我们的角色就是各种名词。我们在初学面向对象的时候，课本上的内容就告诉我们要识别出一个一个的模型，其实，就是让我们识别名词。

识别名词也是很多人对于面向对象的直觉反应。有一些设计方法会先建立数据库表，这种做法本质上也是从识别名词入手的。**我们在战术设计中，要识别的名词包括了实体和值对象。**

什么是实体呢？**实体（Entity）指的是能够通过唯一标识符标识出来的对象。**

我们都知道，在业务处理中，有一类对象会有一定的生命周期。我以电商平台上的订单为例，它会在一次交易的过程中存在，而在它的生命周期中，它的一些属性可能会有变化，比如说，订单的状态刚开始是下单完成，然后在支付之后，变成了已支付，在发货之后就变成了已发货。

但是这个订单始终都是这个订单，因为这个订单有唯一的标识符，也就是订单号，订单号作为它的标识符能将它标识出来。你可以通过订单号查询它的状态，可以修改订单的一些信息，比如，配送的地址。像这种通过唯一标识符标识出来的对象，就是实体。

其实，大多数程序员对于实体并不陌生，因为在各种设计方法中，都有相应的方法识别实体。你甚至可以简单粗暴地将它理解成数据库里存储的对象，虽然这种理解并不完全正确。

还有一类对象称为值对象，它就表示一个值。比如，订单地址，它是由省、市、区和具体住址组成。它同实体的差别在于，它没有标识符。之所以它叫值对象，是因为它表现得像一个值。值对象可能会有很多属性，而要想判断值对象是否相等，我们就要判断这些属性是否相等。对于两个订单地址来说，只有省、市、区和具体住址等多个属性都相同，我们才认为它们是同一个地址。

实体的属性是可以变的，只要标识符不变，它就还是那个实体。但是，值对象的属性却不能变，一旦变了，

www.ixuexi.cf 分享站 用户标记 group_share

它就不再是那个对象，所以，我们会把值对象设置成一个不变的对象。在前面讲函数式编程的不变性时，我给你介绍了不变性的诸多好处，这里也完全适用于值对象。

那你现在应该懂了，**我们为什么要将对象分为实体和值对象？其实主要是为了分出值对象**，也就是把变的对象和不变的对象区分开。在传统的做法中，找出实体是你一定会做的一件事，而在不同的模型中，区分出值对象是我们通常欠缺的考虑。

一方面，我们会把一些值对象当作实体，但其实这种对象并不需要一个标识符；另一方面，也是更重要的，就是很多值对象我们并没有识别出来，比如，很多人会用一个字符串表示电话号码，会用一个 double 类型表示价格，而这些东西其实都应该是一个值对象。

之所以说这里缺少了对象，原因就在于，这里用基本类型是没有行为的。在 DDD 的对象设计中，对象应该是有行为的。比如，价格其实要有精度的限制，计算时要有自己的计算规则。如果不用一个类将它封装起来，这种行为就将散落在代码的各处，变得难以维护。

其实，我们在讨论面向对象的封装时就已经说过了，只有数据的对象是封装没做好的结果，一个好的封装应该是基于行为的。在 DDD 的相关讨论中，经常有人批评所谓的“贫血模型”，说的其实就是这种没有行为的对象。你可以回头复习一下第15讲，我就不在这里赘述了。

关系：聚合和聚合根

选定了角色之后，接下来，我们就该考虑它们的关系了。

在传统的开发中，我们经常会遇到一个难题。比如，如果我有一个订单，它有自己对应的订单项。问题来了，我取订单的时候，该不该把订单项一起取出来呢？取吧，怕一次取出来东西太多；不取吧？要是我用到了，再去一条一条地取，太浪费时间了。

这就是典型的一对多问题，只不过，在其他场景中，主角就变成了各种其他的对象。

不过，这也是一种用技术解决业务问题的典型思路。我们之所以这么纠结，主要就是因为我们考虑问题的出发点是技术，如果我们把考虑问题的出发点放到业务上呢？

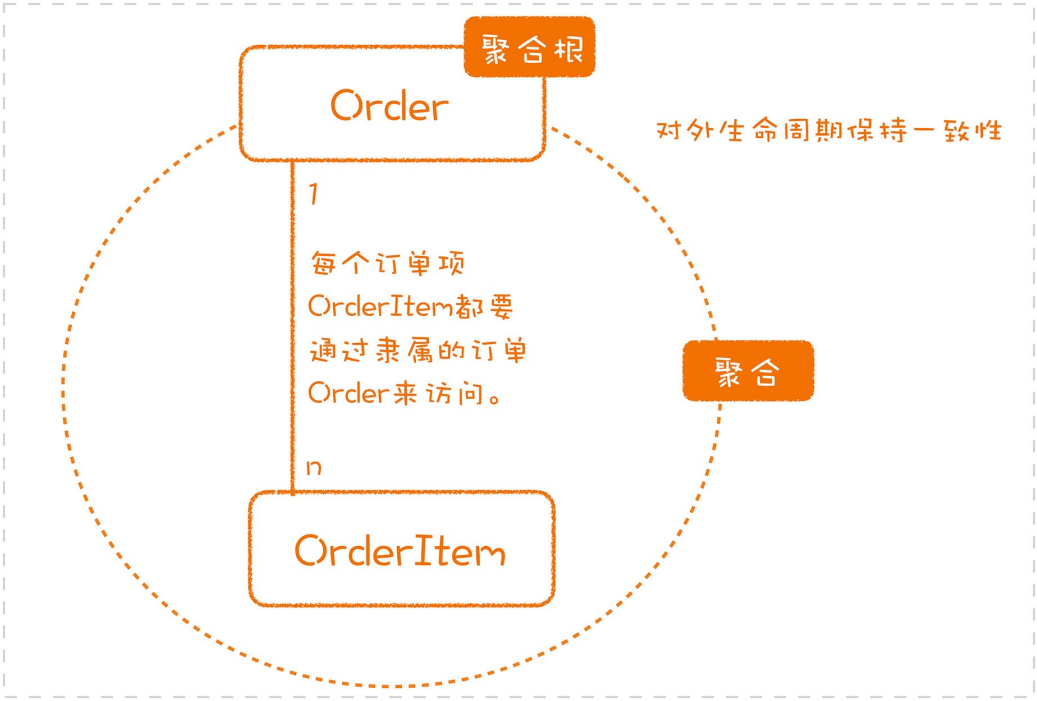
战术设计就给了我们这样一个思考的维度：聚合。**聚合（Aggregate）就是多个实体或值对象的组合，这些对象是什么关系呢？你可以理解为它们要同生共死。**比如，一个订单里有很多个订单项，如果这个订单作废了，这些订单项也就没用了。所以，我们基本上可以把订单和订单项看成一个单元，订单和订单项就是一个聚合。

学习 DDD 时，有人会告诉你，聚合要保证事务（Transaction）一致性。简言之，就是要更新就一起更新，要删除就一起删除。只要你理解了它们是一个整体，你就不难理解为什么这些对象要一起操作了。

不过，一个聚合里可以包含很多个对象，每个对象里还可以继续包含其它的对象，就像一棵大树一层层展开。但重点是，这是一棵树，所以，它只能有一个树根，这个根就是聚合根。

聚合根（Aggregate Root），就是从外部访问这个聚合的起点。我还以上面的订单和订单项为例，在订单和订单项组成的这个聚合里，订单就是聚合根。因为你想访问它们，就要从订单入手，你要通过订单号找到订单，然后，把相关的订单项也一并拿出来。

其实，我们可以把所有的对象都看成是一种聚合。只不过，有一些聚合根下还有其他的对象，有一些没有而已。这样一来，你就有了一个统一的视角看待所有的对象了。所以，我们也可以用统一的标准要求聚合，比如，聚合不能设计得太大。你有没有发现，这其实就是单一职责原则在聚合上的应用。



那如果不同的聚合之间有关系怎么办？比如，我要在订单项里知道到底买了哪个产品，这个时候，我在订单项里保存的不是完整的产品信息，而是产品ID。还记得吗？我们在前面说过，实体是有唯一标识符的。如果需要，我们就可以根据产品 ID 找出产品信息。

有了对于聚合的理解，做设计的时候，我们就要识别出哪些对象可以组成聚合。所以，我们的一对多问题也就不再是问题了：是聚合的，我们可以一次都拿出来；不是聚合的，我们就靠标识符按需提取。**当你纠结于技术时，先想想自己是不是解错了问题。**

互动：工厂、仓库、领域服务、应用服务

我们现在有角色了，也确定关系了。接下来，我们就要安排互动了，也就是说，我们要把故事的来龙去脉讲清楚了。

还记得第27讲的事件风暴吗？我们在其中识别出了事件和动作，而故事的来龙去脉其实就是这些事件和动作。因为有了各种动作，各种角色才能够生动地活跃起来，整个故事才得以展开。

动作的结果会产生出各种事件，也就是**领域事件**，领域事件相当于记录了业务过程中最重要的事情。相对于 DDD 中的其他概念，领域事件加入 DDD 大家庭是比较晚的，但因为其价值所在，它迅速地就成了DDD 中不可或缺的一个重要概念。

因为领域事件是一条很好的主线，可以帮我们梳理出业务上的变化。同时，在如今这个分布式系统此起彼伏的时代，领域事件可以帮助我们让系统达成最终一致的状态。

那各种动作又是什么呢？拿就是我们在写作中常用到的动词。在战术设计中，**领域服务（Domain Service）**就是动词。只不过，它操作的目标是领域对象，更准确地说，它操作的是聚合根。

www.ixuexi.cf 分享站 用户标记 group_share

动词，是我们在学习面向对象中最为缺少的一个环节，很多教材都会教你如何识别名词。在实际编码中，我们会大量地使用像 Handler、Service之类的名字，它们其实就是动词。

你可能会问，按照前面的说法，动作不应该在实体或值对象上吗？确实是这样的，能放到这些对象上的动作固然可以，但是，总会有一些动作不适合放在这些对象上面，比如，要在两个账户之间转账，这个操作牵扯到两个账户，肯定不能放到一个实体类中。这样的动作就可以放到领域服务中。

还有一类动作也比较特殊，就是创建对象的动作。显然，这个时候还没有对象，所以，这一类的动作也要放在领域服务上。这种动作对应的就是**工厂（Factory）**。这个工厂其实就是设计模式中常提到的工厂，有了设计模式的基础之后，你理解起来就容易多了。

需要注意的是，由于聚合的存在，聚合里的各种子对象都要从聚合根创建出来，以便保证二者之间的关联。比如，订单项的产生应该从订单上的订单项工厂方法创建出来。而聚合根本身的产生，就可以由领域服务来扮演工厂的角色。

对于这些领域对象，无论是创建，还是修改，我们都需要有一个地方把变更的结果保存下来，而承担这个职责的就是**仓库（Repository）**。你可以简单地把它理解成持久化操作（当然，在不同的项目中，具体的处理还是会有些差别的）。

其实，很多人熟悉的 CRUD，可以对应成一个一个的领域服务。如果我们用战术设计的做法来表示，应该是这样：

- 创建（Create），从工厂中创建出一个对象，然后，保存到仓库中；
- 查询（Read），通过仓库进行查询；
- 修改（Update），通过仓库找到要修改的对象，修改之后，存回到仓库中；
- 删除（Delete），通过仓库找到要删除的对象，然后，在仓库中删除。

当然，这种简单的映射并不好，没有体现出业务含义，这里只是为了帮助你把已有知识和新知识之间架设起桥梁。

当我们把领域服务构建起来之后，核心的业务逻辑基本就成型了。但要做一个系统，肯定还会有一些杂七杂八的东西，比如，用户要修改一个订单，但首先要保证这个订单是他的。在 DDD中，承载这些内容的就是**应用服务**。

应用服务可以扮演协调者的角色，协调不同的领域对象、领域服务等完成客户端所要求的各种业务动作，所以，也有人把它称之为“工作流服务”。一般来说，一些与业务逻辑无关的内容都会放到应用服务中完成，比如，监控、身份认证等等。说到这里，我们已经说出了应用服务和领域服务之间的区别。

应用服务和领域服务之间最大的区别就在于，领域服务包含业务逻辑，而应用服务不包含。至于哪些东西算是业务逻辑，就要结合具体的项目来看了。

至此，我已经把战术设计这个故事模板给你讲了一遍，DDD 也算完整地讲了一遍了。你现在应该对DDD的各种基础概念之间是个什么关系、如果要做领域驱动设计，要有怎样一个步骤等有一个基本的认识了。

当然，仅凭三讲的篇幅，我们想要完整地理解领域驱动设计几乎是不可能的。但是你现在至少有了一个框

架，当你再去学习 DDD 中那些让人眼花缭乱的知识时，你就不会轻易地迷失了。

Vaughn Vernon 写过两本关于 DDD 的书，是现在市面上比较好的 DDD 学习材料。建议你先阅读《领域驱动设计精粹》，这本书可以帮你快速入门；然后你再看《实现领域驱动设计》，这本书很厚，但讲得要更细致一些。当然，想要真正想学会 DDD，还是需要你在实际项目中进行练习。

总结时刻

今天，我们讲了DDD 中的战术设计，我们把战术设计当作了一个故事模板。让你先去识别角色，也就是找到**实体和值对象**。一个简单的区分就是，能通过唯一标识符识别出来的就是实体，只能通过字段组合才能识别出来的是值对象。

然后我们应该找到角色之间的关系，也就是**聚合**。操作聚合关键点在于找到**聚合根**。当聚合根不存在时，聚合中的对象也就不再有价值了。

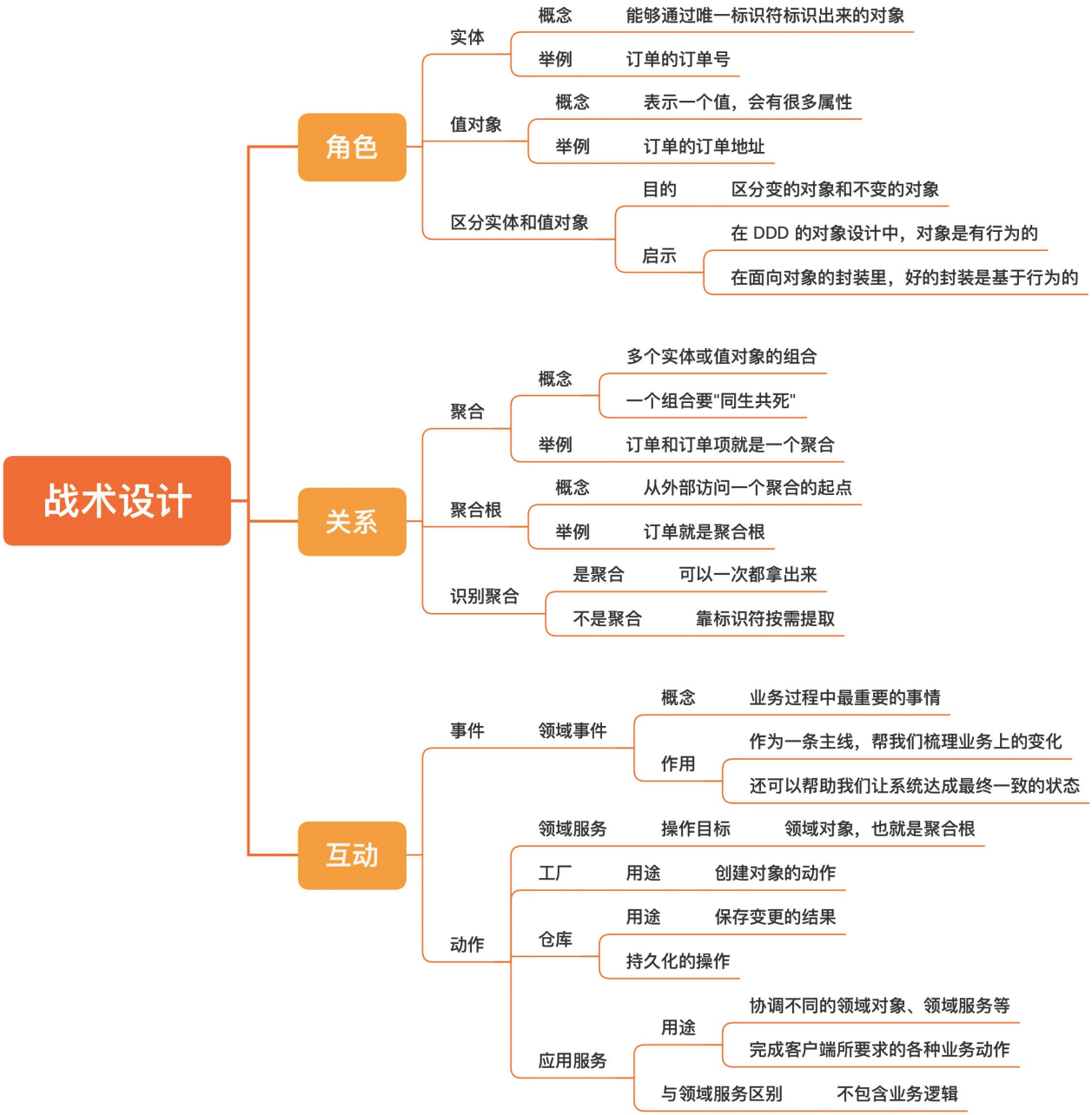
有了角色及其关系，接下来就是找到各种动词，让故事生动起来。这里，我们讲到了动作，也就是**领域服务**，以及动作的结果，也就是**领域事件**，还有创建对象的**工厂**和保存对象的**仓库**。这些内容构成了我们最核心的业务逻辑。一些额外的工作，我们可以放到外围来做，这就是**应用服务**。

通过这几讲关于DDD的学习，你知道了如何识别出各种对象。通过前面设计原则、设计模式的讲解，你知道了如何组织这些对象。至此，我已经把设计相关的主要知识给你讲过一遍了，你现在应该知道如何做设计了。

那现在我们已经有了这样的基础，我们就可以做自己的设计了。从下一讲开始，我们就来体验一下，如何在真实的项目中做设计。

如果今天的内容你只能记住一件事，那请记住：**战术设计，就是按照模板寻找相应的模型。**

www.ixuexi.cf 分享站 用户标记 group_share



极客时间

思考题

最后，我想请你分享一下，你可以按照战术设计的模板，简要地描述一下你的项目吗？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 胖子 2020-08-03 10:08:16
  领域事件可以帮助我们让系统达成最终一致的状态。怎么理解？能举例说明一下吗？

- 骨汤鸡蛋面 2020-08-03 09:43:12
  感觉郑老师最厉害的地方就是讲出了why，而不单是说how。很多文章会说“实体有唯一标识符”，很正确又无用。 只有结合了“是聚合的，我们可以一次都拿出来；不是聚合的，我们就靠标识符按需提取”，我才有了恍然大悟的感觉。

www.ixuexi.cf 分享站 用户标记 group_share